DOCUMENT RESUME

ED 163 916 IR 006 391

AUTHOR	Beard, Marian; And Others
TITLE	Curriculum Information Networks for Computer-Assisted Instruction. Final Report.
INSTITUTION	Stanford Univ., Calif. Inst. for Mathematical Studies in Social Science.
SPONS AGENCY	Navy Personnel Research and Development Center, San Diego, Calif.
REPORT NO	NPRDC-TR-78-18
PUB DATE	Apr 78
CONTRACT	N000123-76-C-1543
NOTE	58p.; Appendix B is minimally legible due to the print quality of the original document
EDRS PRICE	MF-$0.83 HC-$3.50 Plus Postage.
DESCRIPTORS	Branching; *Computer Assisted Instruction; Course Content; Curriculum Development; *Individualized Instruction; *Instructional Design; Interaction; Man Machine Systems; Military Training; *Programed Instruction
IDENTIFIERS	BASIC Instructional Program; Curriculum Information Network

ABSTRACT

This report intended for use by instructors describes research in curriculum design for computer-assisted instruction. The issues and difficulties involved in courseware development are discussed, with particular emphasis on the problems of current "generative" techniques. The Curriculum Information Network (CIN), as used in the BASIC Instructional Program (BIP), is described in detail and its advantages, weaknesses, and possible future development are discussed. Results obtained from analysis of students' experience with BIP are presented. Finally, the relevance of the network idea to Navy Technical Training is discussed and directions for future research are proposed. (Author)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED163916

U S DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

NPRDC TR 78-18

April 1978

CURRICULUM INFORMATION NETWORKS FOR COMPUTER-ASSISTED INSTRUCTION

Marian Beard
Avron V. Barr
Laura Gould
Keith Wescourt

Institute for Mathematical Studies in the Social Sciences
Stanford University
Palo Alto, California 94305

Reviewed by
John D. Ford, Jr.

Approved by
James J. Regan
Technical Director

Navy Personnel Research and Development Center
San Diego, California 92152

IR006391

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPRDC TR 78-18	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): CURRICULUM INFORMATION NETWORKS FOR COMPUTER-ASSISTED INSTRUCTION	5. TYPE OF REPORT & PERIOD COVERED: Final Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Marian Beard, Keith Wescourt, Avron Barr, Laura Gould	8. CONTRACT OR GRANT NUMBER(s): N000123-76-C-1543
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Institute for Mathematical Studies in the Social Sciences, Stanford University, Palo Alto, California 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Z0108-PN.32
11. CONTROLLING OFFICE NAME AND ADDRESS: Navy Personnel Research and Development Center, San Diego, California 92152	12. REPORT DATE: April 1978
	13. NUMBER OF PAGES: 55
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer-Assisted Instruction
Basic Instructional Program (BIP)
Curriculum Information Network
BASIC
Curriculum Design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes research in curriculum design for computer-assisted instruction. The issues and difficulties involved in courseware development are discussed, with particular emphasis on the problems of current "generative" techniques. The Curriculum Information Network, as used in the Basic Instructional Program (BIP), is described in detail and its advantages, weaknesses, and possible future development are discussed. Results obtained from analysis

of students' experience with BIP are presented. Finally, the relevance of the network idea to Navy Technical Training is discussed and directions for future research are proposed.

FOREWORD

This research and development was conducted in response to Navy Decision Coordinating Paper, Education and Training Development (NDCP-Z0108-PN) under subproject Z0108-PN.32, Advanced Computer-Based Systems for Instructional Dialogue, and the sponsorship of the Director, Naval Education and Training (OP-99). The overall objective of the subproject is to develop and evaluate advanced techniques of individualized instruction.

This is the sixth and final in a series of reports dealing with the BASIC (Beginner's All-Purpose Symbolic Instruction Code) Instructional Program (BIP), which is a "tutorial" programming laboratory designed for the student who has no previous training in programming.

Previous reports on the program are concerned with the BIP student manual, supervisor-level manual, conversion into MAINSAIL language, system-level documentation, and student manual revised to reflect the MAINSAIL conversion (Notes 1 through 5 respectively). This report is intended for use by instructors using the BIP system.

The contract monitors were Dr. John D. Fletcher and Dr. James D. Hollan.

J. J. CLARKIN
Commanding Officer

SUMMARY

Problem

Since its inception, CAI has promised to provide individualized instruction. Mechanized individualization of instruction requires the solution to a number of difficult problems. One of the most fundamental of these problems is how to represent the knowledge of a given subject domain in such a way that a CAI system can act "intelligently." In particular, the problem that is addressed here is how to provide an instructional program with an explicit knowledge of the structure of an author-written curriculum.

Objective

The objective of this report is to describe an approach that satisfies the above need: the Curriculum Information Network (CIN) Representation as used in the BASIC Instructional Program (BIP).

Approach

Following a discussion of the issues and problems involved in the development of courseware for both curriculum-based branching CAI and generative CAI, the question of individualized curriculum sequencing is addressed. The advantages and weaknesses of the Curriculum Information Network representation, as used in the BIP system, are discussed.

Findings

The Curriculum Information Network (CIN) shows promise as a fundamental representational structure for CAI curriculum in technical problem-solving areas. The writing of curriculum for the BIP system is straightforward and the CIN provides an extremely flexible structure within which curriculum may be added by nonexperts and through which the content of the course can be tailored to the requirement of different installations. The CIN allows meaningful modeling of the student's progress along the lines of his developing skills rather than just in terms of right and wrong responses. The CIN-based student model is shown to be of value to the individualization of instruction through an intelligent choice of the problem sequence presented to a student. This choice is based both on the model of the subject domain and on the student's past performance.

Conclusions and Recommendations

The Curriculum Information Network is of value as a general method of representing the relationships among the set of skills that combine a curriculum. This approach, as used in the BIP System, seems of particular value to computer-assisted instruction in highly technical skill-related areas.

CONTENTS

	Page
INTRODUCTION	1
Background	1
Intelligent CAI	1
Curriculum-based Branching CAI	2
Generative CAI	4
Problem	8
Objective	8
CURRICULUM INFORMATION NETWORK	9
Description	9
Implementation	11
Strengths and Weaknesses	14
CURRENT IDEAS AND FUTURE EXPLORATIONS	19
Overview	19
Errors and Requests for Help	19
Alternative Student Model	21
Alternative Form for the CIN Itself	21
BASIC Network	22
Representing the Structure of the Skills	27
The Skills Network	28
An Illustration	29
CONCLUSIONS AND RECOMMENDATIONS	31
REFERENCES	33
REFERENCE NOTES	35
GLOSSARY	37
APPENDIX A--THE TECHNIQUE GROUPS AND THE SKILLS	A-0
APPENDIX B--THE LISP NOTATION FOR THE BASIC NETWORK, THE SKILLS STRUCTURE, AND THE SKILLS RELATIONSHIPS	B-0
DISTRIBUTION LIST	

LIST OF FIGURES

		Page
1.	Curriculum-based branching CAI	3
2.	Generative CAI	5
3.	CIN-based CAI	10
4.	The BIP system	13
5.	A piece of the BASIC network	23
6.	The subscripted variable portion of the BASIC network	25
7.	The PRINTST portion of the BASIC network	26

INTRODUCTION

Background

Intelligent CAI

Much current research on CAI has stressed the development of systems that exhibit some form of intelligent behavior (Barr, 1976). In the context of CAI, "intelligent behavior" implies that:

1. The program can make dynamic instructional decisions based on the student's previous interactions.

2. The program can provide error correction or problem-solving help specifically relevant to the student's input.

3. The dialogue can be conducted in some reasonable subset of English.

4. The program can evaluate important aspects of the student's responses by using procedures that "know about" the subject matter, rather than by accessing the author's prepared list of expected right and wrong responses.

Two major aspects of courseware have been investigated via intelligent CAI. First, the student-machine interface has been broadened to use English (Brown, Burton, & Bell, 1974; Carbonell, 1970), computer-generated audio (Atkinson, Fletcher, Lindsay, Campbell, & Barr, 1973; Sanders, Benbassat, & Smith, 1976; Van Campen, 1970), voice recognition (Danforth, Rogosa, & Suppes, 1974), and graphics (Bork, 1975). Second, research in answer checking and analysis has used many different intelligent routines: Proof checkers (Goldberg, 1973; Smith, Graves, Blaine, & Marinov, 1975), a REDUCE-like algebraic simplifier (Kimball, 1973), a circuit simulation that knows how to debug itself (Brown et al., 1974), and program-analyzing programs (Goldstein, 1975; Ruth, 1974). One course under development at Stanford uses a natural language parser, an algebraic simplifier, and a sophisticated proof checker to converse with the student about set theoretic proofs in informal English (Smith & Blaine, 1976).

Since its inception, CAI has promised to provide "individualized" instruction, one aspect of which is sequencing the curriculum material optimally for each student. Our work in Intelligent CAI has focused on the issue of the "representation" of the subject domain (which is also a fundamental concern of current research in cognitive psychology and artificial intelligence). The goal is to provide a representation of the subject matter that is sufficient for individualized presentation of the material. A consideration of the different "representational poles" in vogue in CAI will give a perspective on the capabilities of the Curriculum Information Network representation.

We note first that individualization in CAI can have many different goals. Some systems are designed with informal, flexible student-program dialogue as the most important feature; such systems rely heavily on powerful natural-language processors to analyze the student's input and respond appropriately to his questions (Brown et al., 1974; Collins, Passafiume, Gould, & Carbonell, 1973). Other systems are designed to respond to students' errors or requests for help in different ways; by maintaining a record of the concepts previously mastered by each student, these systems can decide whether

to "give away" the correct answer, to present a partial answer or a leading question as a hint, or to perform itself some of the more trivial work for the student (Koffman & Blount, 1975). Similarly, such a record (or student history or model) can be used to locate information directly related to facts that the student already knows, so that the program can build on his knowledge most effectively. Another aspect of individualization is the matter of problem selection as the student progresses through a nonfixed sequence of curriculum, where the goal is to present a problem appropriate to his current state of understanding (or confusion). It is this last aspect that has concerned us most in our work, and the following comparisons among various styles of CAI is therefore addressed primarily to the question of individualized curriculum sequencing.

Curriculum-based Branching CAI

The most common style of CAI courseware now being written, which we will call "curriculum-based branching CAI," is an automated presentation of a curriculum written by a human author. The author, being knowledgeable in the subject matter, has in mind a clear organization of the interrelations among the specific facts of that subject, an implicit understanding of the dependency of one concept on another, and a plan for the development of new skills. His personal organization of the discrete elements results in a structured curriculum, consisting of lessons or problems presented in a sequence he considers to be optimal in some sense for his model of his students. This structure is like that of a textbook, established in advance of interaction with the student, but potentially superior to a textbook in that the author builds branching decisions into the program, providing some degree of individualization. His subdivisions of the curriculum and the branching criteria he specifies constitute the author's representation of the subject matter in this traditional CAI style.

Figure 1 illustrates the basic mechanism of individualization in branching CAI. The instructor-supplied curriculum consists of lessons, exercises, multiple-choice questions, tests, problems, or problem forms (to be filled in by the instructional program). The student starts at the beginning, and a record of each curriculum element he sees is kept in a student history of some form. Problem-selection rules of the following style are then used to choose the next curriculum element on the basis of the student's history:

> If correct then go to problem 8, otherwise problem 6.
> If answer is 9 instead of 6 then student is wrong and
> go to exercise 13.
> If percent correct < 75 then go to review lesson III-R1.
> If percent correct > 90 then skip next lesson.

The point here is that the branching rules are based on the curriculum elements themselves, which is the only aspect of the curriculum that the program "knows about." These rules can be built into the structure of the curriculum (e.g., as standard decision procedures to be followed at the end of each lesson), or supplied explicitly (e.g., within particular problems, where certain responses can be specially treated); the most typical case is some combination of both. In most cases, however, the end result of this kind of individualization has been either that fast students are allowed to detour around blocks of material, or slow students are given remedial lessons.

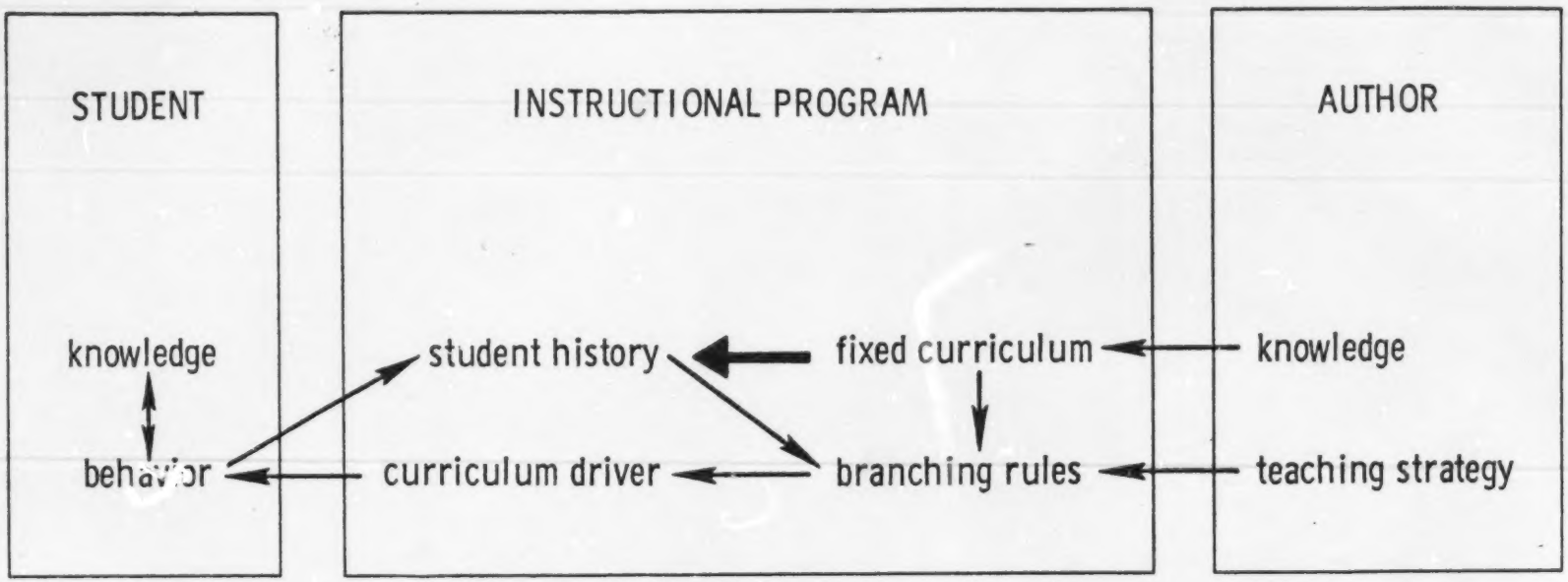

Figure 1. Curriculum-based branching CAI.

An example of this style of CAI is the AID course developed at Stanford (Friend, 1973). This was a large-scale course, with 100 hours of curriculum material to teach the AID (Algebraic Interpretive Dialogue) programming language at the introductory undergraduate level. This course was used in colleges and junior colleges as a successful introduction to computer programming (Atkinson et al., 1973; Beard, Lorton, Searle, & Atkinson, 1973; Friend, 1975). However, it was a linear, "lesson-oriented" CAI program that used branching decisions like those listed above. One limitation of the course was that it could not provide useful instruction during the problem-solving activity itself. After working through lesson segments on such topics as syntax and expressions, the student would be assigned a problem to solve in AID. It was necessary for him to leave the instructional program, to call up a separate AID interpretor, to perform the required programming task, and to return to the instructional program with an answer. As he developed his program directly with the AID interpreter, his only source of assistance was the minimally informative error messages provided.

More importantly, the AID course was found to be an inadequate vehicle for more precise investigations of individualization of instruction because of the linear organization of its curriculum. The course consisted of a large set of ordered lessons, reviews, and tests; and a student's progress from one segment to the next was determined by his score on the previous segment. A high score would lead to an "extra credit" lesson on the same concepts, while a low score would be followed by a review lesson. It became clear that this decision scheme, based on total lesson scores, was reasonably effective in providing instruction and programming practice. However, since it dealt with rather large segments of the curriculum, the precision with which it could respond to different students' difficulty with specific concepts was minimal. When allowed to select a lesson, students almost invariably chose to proceed to the next (numbered) lesson or to review just-completed material. For example, even if a student recognized that he was confused about initializing and incrementing, his only choice was to request the review lesson on loops. Because of its limited ability to characterize individual students' knowledge of specific skills, and its inability to relate those skills to the curriculum beyond determining a ratio of problems correct to problems attempted, all students covered the same concepts in roughly the same order, with slight differences in the amount of review. (See Beard et al., 1973, for details.)

Generative CAI

At the opposite pole of curriculum structure are "generative" CAI programs, which do not use an author-written curriculum at all. This type of course generates problem statements and solutions by retrieving information from a complete, internal representation of the knowledge underlying the subject domain.

An important goal of generative CAI is to incorporate the instructor's knowledge of the material into the instructional program itself (see Figure 2). The program has a data base containing all the "facts" the student is to learn, as well as the inference procedures that the instructor uses to access those facts. It also uses the instructor's pedagogical knowledge of how to present facts that the student doesn't know in a way that will be understandable. Its student model is more like a representation of what the student knows, not just what his behavior on previous problems has been.

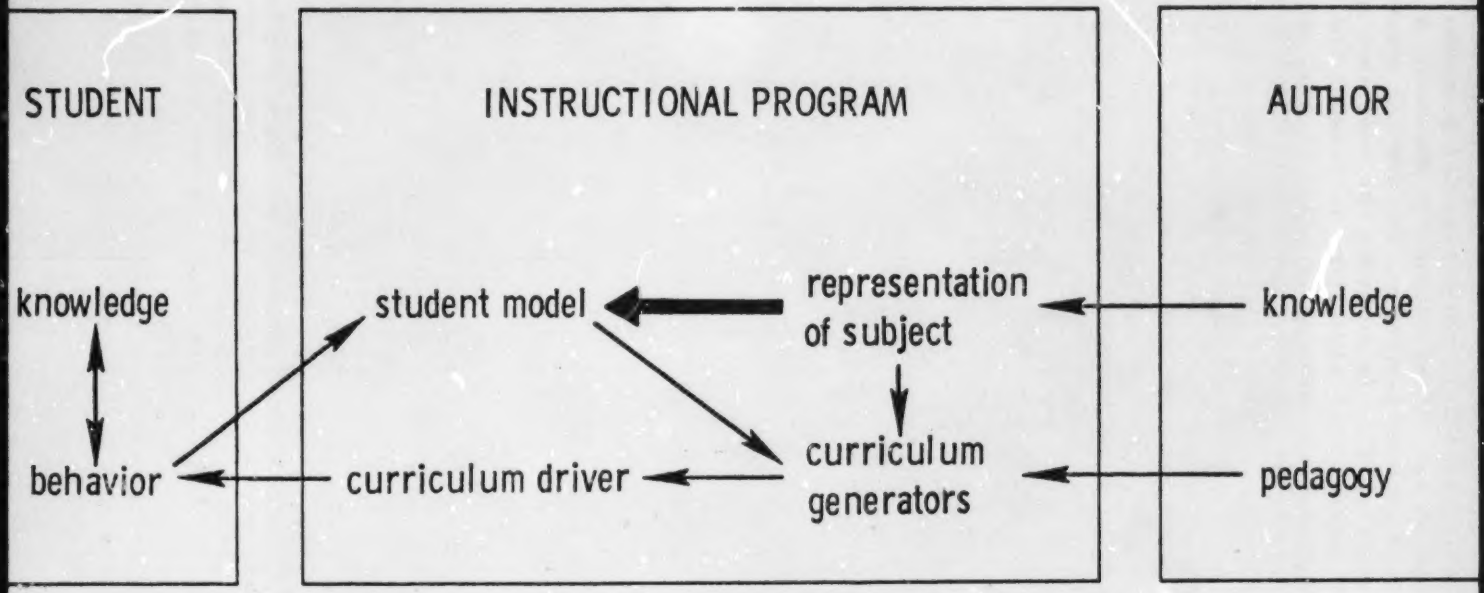

Figure 2. Generative CAI.

There are several examples of excellent generative CAI programs in various subject areas (e.g., Brown et al., 1974; Grignetti, Hausmann, & Gould, 1975). For factual subjects such as geography, the SCHOLAR system (Carbonell, 1970; Collins & Grignetti, 1975) is probably the best such model. SCHOLAR uses question-and-answer construction algorithms to present the material in the data base to the student. Carbonell's original goal was to build a program capable of generating questions and answers in a nonalgorithmic subject area, where the information represented in the program was ill-defined verbal knowledge rather than a well-defined subject like mathematics. More recent versions and applications of the program have built on this original idea, incorporating features such as visual display and expanding the question-and-answer generating capabilities into other subject areas such as on-line text editing (exemplifying procedural knowledge) and climatic effects on agriculture (causal knowledge and hypothesis formation). In terms of the different aspects of individualization mentioned above, the goal of these applications has been, generally speaking, to respond more appropriately to the student in the immediate situation. That is, recent work with SCHOLAR has aimed at making the program more effectively able to generate examples based on the student's current error, or a next question based on his most recent hypothesis. In both cases, reasoning mechanisms have been used to infer the student's intent. These mechanisms access the "event memory" or history of the student's interaction stored within the program and, in this way, the SCHOLAR system and our work with the Curriculum Information Network are similar.

The major difference between the two projects is that SCHOLAR does not focus on the complexities of problem selection. The semantic network, in which all of the information about the subject is stored, is organized as an outline of topics and subtopics, each with an "I-tag" indicating its importance. In the version of SCHOLAR that dealt with geography, "problem selection" worked as follows: Within time constraints, the program discussed the information under the current topic down to a pre-specified level of importance. When the allotted time expired, the program backed up to another high-level topic and began again to ask questions, to provide related information, and to give review down through the subtopics in the order of their importance. The version of SCHOLAR that taught the use of the on-line editor proceeded through a set of lessons; hints and error correction were generated dynamically in response to the student's input, but the overall path through the "curriculum" was fixed.

Although the recent work with SCHOLAR has moved beyond the strictly factual subject matter of geography, the dialogue between the program and the student is still characterized by one question at a time. The NLS-SCHOLAR system allowed the student access to the text editor itself, but the hands-on manipulation was limited to performing a very limited sequence of editing changes. In general, SCHOLAR does not deal with problem solving; the student is not given a complex task to complete. Thus, it has not been necessary for SCHOLAR to concern itself with the issue of problem selection, the matter of determining what area of the curriculum the student needs to work with at each stage of his progress.

To illustrate the difficulties involved in producing generative courseware in complex problem-solving subjects, consider Koffman's pioneering course in machine language programming (Koffman & Blount, 1975). This system (MALT) uses a set of programming primitives to generate programming tasks (by combining primitives), which it can both present to the student (in English) and solve with a program (since it can solve all of the primitive tasks). One advantage of this approach is that the system can generate and solve a large variety of problems. Another, and perhaps more important in our view, is the system's ability to present increasingly difficult problems as a function of each student's competence and prior experience. Koffman (1972) describes his "intelligent CAI monitor" as:

> . . . centered around the use of a student model (summary of a student's past performance) and a concept tree, which indicates the prerequisite structure of the course. As the system gains experience with a particular student, it updates his model and establishes to what extent he prefers to advance quickly to new material or build a solid foundation before going on. Based on its knowledge of the student and his past performance, it decides at which plateau of the concept tree the student should be working. All concepts of this plateau are then evaluated with respect to factors such as recency of use, change in state of knowledge during last interaction, current state of knowledge, tendency to increase or decrease his state of knowledge, and relevance to other course concepts. The highest scoring concept is selected, a problem suitable for his experience level is generated, and its solution is monitored.

The disadvantages of the MALT system are that it requires the student to follow its own sequence of primitive steps in solving the problem, and that its problem statements are lean, and, in our opinion, unmotivating. For example:

> Your problem is to write a program which will:
>
> Read in 10 (octal) 1-digit numbers and store their values starting in register 205.
>
> Here are the sub-problems for the 1st line:
>
> 1. Initialize a pointer to register 205.
> 2. Initialize a counter with the value of -10 (octal).
> 3. Read a digit and mask out the left 9 bits.
> 4. Store it away using the pointer.
> 5. Update the pointer.
> 6. Update the counter and if it's not zero, jump back to start of loop.

Thus, although the problem-selection process seems clearly to add to the MALT system's ability to individualize the sequence of instruction, we feel that the system suffers from the dry style of its problem statements.

The advantages for individualization of generative CAI over fixed-branching courseware are considerable: The generative program can provide instruction and/or information in precisely the areas needed by the student. All decisions about what material to present can be made dynamically, based on the student's progress with the subject matter, rather than on a predetermined sequence of material. Ideally, the program has access to the same information that makes the human author a subject matter expert, and this information can be made available to the student much more flexibly than is possible in author-generated CAI. In particular, the model of the student's state of knowledge is based on the structure of the subject itself (e.g., the student has covered the material on rivers in Brazil) rather than on the structure of the author's curriculum design as reflected in his branching specifications, which are typically triggered by correct/wrong response counters.

The major disadvantage of generative CAI in technical subjects is that generated questions limit the student's hands-on interactions with the subject matter, while generated problems are not very interesting.

Problem

In technical subjects, development of skills requires the integration of facts, not just their memorization, and the organization of instructional material is crucial for effective instruction in these areas. As the curriculum becomes more complex, involving the interrelations of many facts, the author's ability to present it in a format that facilitates assimilation and integration becomes more important. At the same time, however, using counters reflecting performance on questions or lessons to keep track of the student's progress through the curriculum provides a less adequate model of his acquisition of knowledge. An approach to individualized CAI is needed that incorporates the positive aspects of both author-written and generative CAI.

Objective

The objective of this report is to describe an approach to individualized CAI that meets the above requirements: The Curriculum Information Network (CIN) Representation, as used in the BASIC Instructional Program (BIP) system.

CURRICULUM INFORMATION NETWORK

Description

In this approach, problems are selected on an individual basis using a representation of the "meaning" of the curriculum, called the Curriculum Information Network (CIN). The CIN is intended to provide the instructional program with an explicit knowledge of the structure of an author-written curriculum. It contains the interrelations between the problems that the author would have used implicitly in determining his "branching" schemes. It allows meaningful modelling of the student's progress along the lines of his developing skills, not just his history of right and wrong responses on the problems, without sacrificing the motivational advantages of human organization of the curriculum material. The instructional program can monitor the student's progress on these skills, and choose the next problem with an appropriate group of new skills. An intermediate step is introduced between recording the student's history and selecting his next problem: thus, the network becomes a model of the student's state of knowledge, since it has an estimate of his ability in the relevant skills, not just a record of his performance on the problems he has completed. Branching decisions are based on this model instead of being determined simply by the student's success/failure history on the problems he has completed, as shown in Figure 3.

In curriculum-based branching, simple problems often focus on one particular skill that, by itself, the student may have mastered. On the other hand, complicated problems may involve a large number of different skills, some of which are impossible for the student at his current level of learning. Neither of these experiences is likely to contribute to his progress. The CIN approach attempts to avoid boring the student or frustrating him, by selecting problems whose requirements are very close to his current abilities--neither annoyingly trivial nor impossibly complicated.

Generative programs also do a better job of presenting appropriately difficult problems, but their context and wording (produced by the program, not by a human) are typically very boring. To the student, the problems seem to be mechanical exercises rather than challenging human puzzles. The CIN makes it possible to present much more interesting curriculum without sacrificing the program's ability to estimate specifically the aspects of the subject area in which a student needs work.

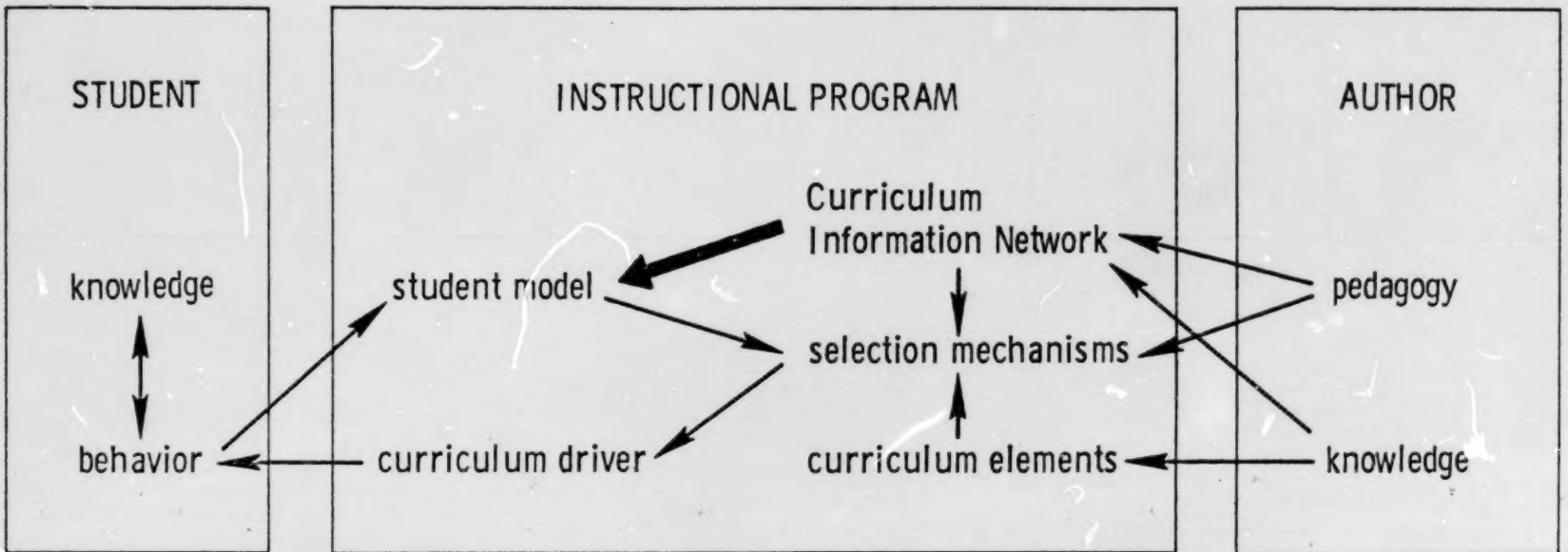

Figure 3. CIN-based CAI.

The following is an example of a programming problem taken from our CAI course in programming, discussed in the next section.

> On the first day of Christmas, someone's true love sent him/her a partridge in a pear tree (one gift on the first day). On the second day, the true love sent two turtle doves in addition to another partridge (three gifts on the second day). This continued through the 12th day, when the true love sent 12 lords, 11 ladies, 10 drummers, . . . all the way to yet another partridge.
>
> Write a program that computes and prints the number of gifts sent on that 12th day. (This is not the same as the total number of gifts sent throughout all 12 days--just the number sent on that single 12th day.)
>
> The skills that describe this task are:
>
> Initialize numeric variable (not counter) to literal value
> FOR . NEXT loop with literal as final value
> Accumulate successive values into numeric variable
> Multiple print: string literal, numeric variable

Problems are selected on the basis of the student's performance on the skills underlying the curriculum; thus, this problem is presented either when the student is ready to learn about FOR . NEXT loops that accumulate a sum, or after he has had difficulty with such skills in a different problem, and therefore needs more work on those skills.

Computer-assisted instruction has long promised to present an individualized sequence of curriculum material, but in most cases this has meant only that "fast" students are allowed to detour around blocks of curriculum, or that "slow" students are given sets of remedial exercises. By describing the curriculum in terms of the skills on which the student should demonstrate competence, and by selecting problems on the basis of individual achievement and difficulties on those skills, we believe that a truly individualized instructional system can be built. We have explored this approach to "tutorial" CAI in programming, and we believe the curriculum structure we have developed should be applicable in many other subject areas that involve identifiable skills and that require the student to use those skills in different contexts and combinations.

Implementation

We discuss in this section an implementation of a Curriculum Information Network in a fully operational CAI course. Our experience over the past three years with the BASIC Instructional Program (BIP) has given us some useful insights into the power and limitations of the CIN (in the context of teaching computer programming), which we believe are relevant to CAI courses in other technical, problem-solving types of subjects.

The development of the BIP system has been supported by the Office of Naval Research, the Advanced Research Projects Agency, and the Navy Personnel Research and Development Center. The course is fully described by Barr, Beard, and Atkinson (1975a, b). It is designed to introduce students to programming in the BASIC language, almost exclusively through guided hands-on practice in writing and running programs. Figure 4 illustrates the relationships among the parts of the entire system. Using the information in the CIN and the student model, the task-selection procedures present the student with a problem ("task") to solve, typically of the form "Write a program that . . ." As he types his program, the interpreter presents specially-designed instructional messages when errors occur. The student also has access to hints (both graphic and text) and debugging facilities, and he may execute the stored "model solution" at any time to see how his own program is expected to behave. The solution checker evaluates his program by comparing its output to that of the model solution; if his program is not acceptable, he may choose either to leave the task at that time or to continue working on his program. When he leaves, either by "quitting" at this point or by successfully completing an acceptable program, the "Post Task Interview" asks him to evaluate his own competence on the skills involved in the task. The student model is updated to reflect the student's success in the task (or his choosing to quit) and his responses to the Post Task Interview. The essential features of the task-selection and model-updating processes will be described more fully below.

The distinctive feature of the CIN-based curriculum is the existence of the problems as an unstructured pool of curriculum elements avaiable for presentation to the student; the actual sequence in which he sees the tasks depends on the authors' "advance" strategy only when a dynamic decision cannot be made. The tasks in BIP represent widely varying degrees of difficulty, and it is certainly possible to order them in any one of a number of pedagogically reasonable fixed sequences. This kind of sequencing resembles the order of chapters in a textbook, a reasonable path that one might take, but suited only to some hypothetical average student. The purpose of the CIN and its associated student model and selection strategy is to use the information resulting from a particular student's interaction to order the presentation of tasks; the system is designed to respond to individual differences in ways that are much too complex for the author to have anticipated, much less specified, fully beforehand for all students.

The most important part of the CIN for purposes of task selection (as opposed to helping the student solve the problem, for example) is the relationship between each task and its skills. In BIP, the skills are specific descriptions of particular programming behaviors like "print a string literal" or "initialize a counter variable." Each task is described by the list of skills that are required in the solution program, and the student model reflects progress on each of the skills, not on the tasks. We feel that this "skill history" is crucial to determining the most appropriate next task for a student at any point in the curriculum. The skills represent knowledge about (or mastery of) aspects of programming itself, and a given task is selected because it embodies some of those aspects of programming that the student is currently ready to deal with, whether for remediation or new learning.

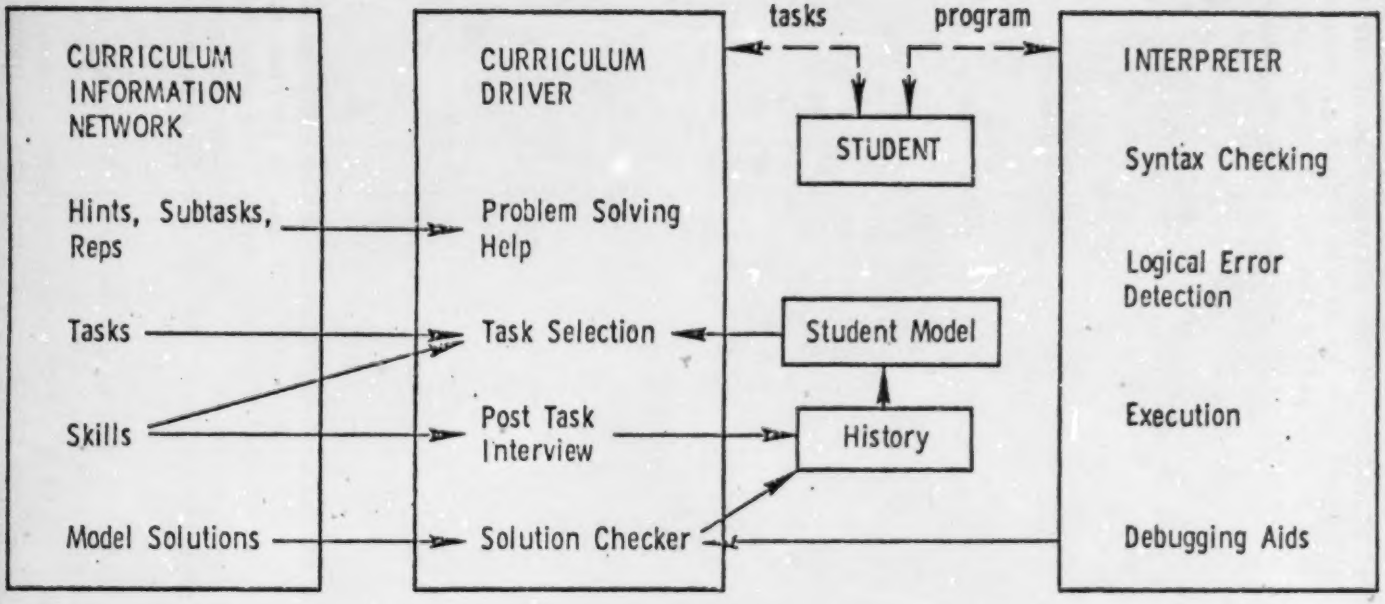

Figure 4. The BIP system.

In addition to describing the tasks in terms of the skills they require, we impose a general structure on the presentation of curriculum by grouping the skills into techniques such as simple printing, assignment, conditional branching, etc. (All of the skills and the technique groups are listed in Appendix A.) BIP's task-selection process works as follows: Starting at the lowest (least complex) technique (i.e., No. 1), all the skills grouped at that level are compared to the student model; if any skills are considered to "need further work," a task using some of those skills is sought. (If no such "needs-work" skills are found, the skills in the next higher technique group are compared to the student model and the process continues.) If a task is found that uses some skills needing work, and that does not require any skills at a higher level, it is selected and the process ends. If no task can be found at a suitably low level, the process moves to the next higher technique and reapplies the criteria. Thus, the current mechanism in BIP relies on the technique hierarchy for its overall sequencing: Skills at low levels will be presented before those at higher levels, which simply means that easier tasks will precede more difficult tasks.

Within that overall framework, however, the choice of tasks can be quite wide. A skill is considered to be "mastered" if a student has completed a task using that skill successfully, both by passing the solution checker and by indicating in the Post Task Interview that he understood the use of the skill in the context of the task. A student who had no difficulty with BIP's curriculum, always passing the solution checker and indicating his confidence in his own understanding, would follow a completely predictable path through the techniques. More typically, students have trouble with various concepts and programming structures, and therefore follow widely divergent paths, since BIP's developing model of their progress indicates different patterns of mastery of the skills. Some remain at a given technique level for longer periods of time, receiving more work on certain skills; others bounce down to lower levels for specific remediation on other skills, etc. The variety of resulting sequences resembles the variety of experience provided by a strictly generative CAI course, since progress is measured by success within the representation of programming knowledge rather than on a set of lessons.

Strengths and Weaknesses

Although we are pleased with CIN's level of responsiveness to individual students' needs, our experience with BIP has shown that the CIN representation has some weaknesses. We will describe some of the side benefits of the CIN scheme, and then discuss some of the drawbacks as background for the explorations currently under way and planned for the future, presented in the next section.

One useful feature of BIP as it now operates is the information it provides about its own processes. The most important process is, of course, that of task selection, and the program generates its own record of failures in seeking the appropriate task. If no task can be found that requires some "needs-work" skills without requiring any skills at higher levels, this failure is recorded as a "hole" in the curriculum. The information describing the hole includes the skills that were being sought, the technique level that the process had reached, and the reason for the failure--either no task with "needs-work" skills or none with skills at suitably low levels. About 20 tasks were added in one major curriculum revision, and another, based almost

entirely on the holes recorded since that time, will be done before the developments described in the next section are evaluated. By comparison, to determine the content weaknesses of his curriculum in traditional curriculum-based branching CAI, the author must assemble and analyze a variety of indirect data (such as test performance). Having the CIN-based selection process record its own failures gives immediate, direct, automatically-generated data that tells exactly what should be added to the curriculum.

Other data provide direct information on the nature of both student and program performance. The student model is itself a reflection of the student's progress--not through the tasks but through the specific aspects of programming represented in the CIN. The patterns of concentration on certain skills (and the lack of emphasis on others) indicates those areas of the subject matter that may be receiving too much (or too little) attention. Such weaknesses can be remedied by changing either the content of the curriculum, the technique groupings, or the task-selection process. All of these approaches, singly and in combination, have been used during the evolution of the program to improve the balance of skills presented. We emphasize that the CIN-based design allows us to experiment with different mechanisms for selecting among tasks without changing the tasks themselves. Again, the contrast with fixed-branching CA is strong: Since such curriculums are conceived of as a whole, it is almost impossible to change the way in which the problems are presented without changing the whole structure (usually lessons) in which they are imbedded.

Recently the BIP system was made available to the Naval Academy for an operational evaluation in a Navy setting. Thorough analysis of all aspects of the operation is not within the scope of this report, but we draw on examples from the data to illustrate a few specific points about the strengths and weaknesses of BIP's CIN. In general, the midshipmen's experience with BIP was favorable; specifically with respect to the sequence of tasks selected, the CIN seems to have provided considerable individualization in response to students' different rates of progress.

One quick measure of the "goodness" of a sequence of tasks is the degree to which the student progressed smoothly through the technique levels, which generally reflect the increasing complexity and difficulty of the tasks. Ideally, the sequence should involve no large jumps, either upward or downward, in complexity. Among the 534 tasks selected by BIP's mechanism (i.e., not specifically requested by the student himself), 71 instances occurred in which the technique level changed by more than one, either upward or downward. (Only five of the 16 students experienced more than five such "breaks" in the progression.) The causes of the "breaks" include: (1) simple failure of the process to perform in a pedagogically correct way--a design problem whose roots will be discussed further in this section, (2) student requests for more work on skills at low levels--requests which are always honored if a sufficiently easy task is available, and (3) mastery of certain skills in the context of student-selected tasks. (If a student chooses to select a few tasks by name, the next task selected by BIP is often at a technique level three or four higher than that of the BIP selection that preceded the student's self-guided sequence.) Since we have not yet analyzed the students' protocols in detail, we have not identified all of the "breaks" that were caused by students' requests for specific tasks or for more work on specific skills. The failure rate of our CIN implementation due entirely to weaknesses in the design is actually lower than 71/534. We feel that this ratio is a satisfying confirmation of the overall usefulness of the CIN implementation.

The major weakness of the CIN as we use it is the reliance on the technique structure as the governing hierarchy for the task-selection process. The skills at a given level are not necessarily analogous to each other in the sense of dealing with similar concepts or similar programming semantics--they are just thought to be similarly difficult to use. The sequence of tasks that results from the technique-based task-selection process occasionally appears to be arbitrary with respect to the content of the problems, even though the progression of difficulty appears reasonable. The techniques also add little to BIP's ability to make useful inferences about the different contexts in which a given skill might appear, differences that might contribute to a student's difficulty with a supposedly well-learned skill. The hierarchy of techniques was originally intended to provide an overall guide for the sequencing of tasks, and we feel that it has succeeded well in this general goal. But the technique structure does not specify the relationships among skills precisely enough to reflect accurately the portion of programming knowledge that is covered by our curriculum. We have observed a number of specific symptoms of this general weakness which the following examples illustrate.

One student's record shows a sequence of tasks at technique levels 10, 11, 13, 15, then suddenly dropping back to a task at technique 6. More important than the drop in numbers was the difference between the two adjacent tasks. The student had successfully completed task BACKARRAY, which requires a program that obtains an array of words from the user, and prints the array in backwards order. The next task selected by BIP was INPUTSUM, which requires a program that gets two numbers from the user and prints their sum. Obviously, INPUTSUM is too easy for a student who successfully handled BACKARRAY. BIP selected the easier task for the following reasons: The student had quit a difficult task at technique 13 (choosing to leave after failing the solution checker); one of the skills in that task was Skill 29, which appears in a much lower technique and which the student had used successfully in one earlier task. After a quit, the counter representing each of the skills in the task is decremented. He chose the BACKARRAY task himself, and passed it, but since it did not require Skill 29, that skill's counter was at zero, indicating that he needed more work on it. Thus, when the task-selection process climbed through the techniques, it identified 29 as a skill to be sought in the next task, and then found INPUTSUM, which requires that skill, at technique 6.

This illustration is fairly typical of the worst failures of the task-selection procedures to locate an appropriate next task. Students who have progressed very rapidly up to the time that they quit a difficult task are particularly vulnerable to this kind of "drop," since they are much more likely than slower students to have seen many skills only once previously. The student model as it is now implemented often does not accurately reflect the student's knowledge of the skills. A more complex model of the student is required (see below). Also, the usually successful strategy of beginning the search for "needs work" skills at the lowest technique clearly is not adequate in all cases.

A second illustration will illuminate additional issues that need to be considered in designing the CIN implementation. This second student quit task PITCHER, a fairly difficult task at technique level 12, and was next

presented with SREAD, a much easier task at technique 5. (He had previously succeeded with a few tasks at higher levels, so SREAD definitely appears to have been too easy.) The cause of this drastic drop was, again, a single skill (16) which, when decremented after he quit PITCHER, was considered to need more work. Skill 16 occurs at a fairly low technique (5), and is required in task SREAD, which was therefore presented next. One interesting feature of the PITCHER-SREAD sequence is that only two of the skills required in PITCHER were in the MUST set of "needs work" skills; yet it was a skill not in the MUST set (namely Skill 16) whose decrementing led to the too-easy next task. This apparent irrationality results from the fact that the MUST set is established when a new task is requested, not when the current task is completed. Thus, in this case, the skills in the MUST set at the time PITCHER was selected were ignored when SREAD was selected (because they all appear at higher levels). A possible improvement might be to establish the MUST set after the completion of each task, so that the requirements for the next task would be more similar to that last task than is currently the case. Implementing this scheme might, however, make it too difficult for a student to move into different kinds of tasks, especially in cases of failure when more drastic moves might be most appropriate.

Other issues related to the application of the CIN to task selection and updating the student model have arisen, some of which we hope to deal with in the future. For example, how should the model reflect a student's difficulty with a task, or his choosing to leave the task without even attempting to write a solution program? Our assumptions have almost always given the student the benefit of the doubt. We choose to ignore "difficulty"--the tremendous amount of time spent in a task, repeated failures to pass the solution checker, etc. If a student writes an acceptable program for a given task, we assume that he has learned the material presented in that task, and we upgrade its skills as though he had passed on his first attempt. Similarly, if he chooses to leave the task without yet having failed the solution checker, we make no judgment about his mastery or difficulty with the skills involved; their counters are neither incremented nor decremented. Our reasoning is that a student should be allowed to avoid tasks that are not interesting to him. The next task selected should have a very similar group of skills, so we have not worried about students missing significant portions of the curriculum. (Of course, exercising this option can be taken to an extreme, and in some controlled studies we have disabled it.)

Another question deals with the parameters of the task-selection process. Given that we have assembled a MUST set of skills, should we present a task that has as many of those skills as possible (as is now the case), or should the proportion of MUST skills be somehow related to the student's current competence? For example, it might be more reasonable to find the task with only one MUST skill early in the student's experience with the course, or after a succession of "quit" situations. As he gains confidence and competence, the number of allowable MUST skills could increase, theoretically resulting in an increasing rate of increasing difficulty. A likely problem with this scheme lies in the relative nonredundancy of the curriculum at the higher levels: Just as PITCHER required only two of the MUST skills, it may often be impossible to find tasks with as many MUST skills as we would like, and we may have to increase the size of the curriculum in order to test this scheme.

These comments, we hope, have given a sense of the complexities and difficulties involved in designing and using a network of information to improve the program's ability to choose appropriate tasks on the basis of a student's current performance, confidence, and expressed choices. The next section describes some of the ways in which we have experimented with alternative structures and processes.

CURRENT IDEAS AND FUTURE EXPLORATIONS

Overview

In this section, we discuss a few relatively minor additions that we have considered making to BIP's CIN, and two modifications whose effects we expect will be more significant. The additions involve taking advantage of students' errors and requests for help in improving the accuracy of the student model, whereas the new modifications will constitute major changes to the student model and the network itself.

Errors and Requests for Help

A student's errors during programming or his requests for help on a task are indications of his difficulty in understanding some concept or dealing with some skill. (They may also reflect differences in students' learning styles.) With sufficient machinery in the instructional program, it should be possible to infer from his interaction with the system some specific information as to the skills that are giving him the most trouble, and to reflect this information in the student model. While we do not expect BIP to become capable of formal program analysis, we have developed some less ambitious additions to the program that should enhance the intelligence of its behavior. The first of these would improve the usefulness of the error messages delivered by the program, and the second would use specific errors and requests for help to update the student model.

The CIN in BIP currently includes only information about the content of the tasks and the groupings of the skills into techniques. Information might be added that could relate students' specific errors to the curriculum in a way that would improve the relevance of the interpreter's error messages. The interpreter gives three kinds of messages: (1) Syntax errors are detected as the student types a line that cannot be parsed, (2) preexecution structural errors (e.g., a branch to a nonexistent line) are detected just before the program is run, and (3) runtime errors are detected when an unexecutable statement (e.g., using the value of an unassigned variable) is encountered. The error messages were carefully designed to be as instructive as possible (Barr et al., 1975b), and often include pointers to the erroneous part of the student's line or program. However, the interpreter is not directly linked to the curriculum in any way, and some links might be appropriate.

For example, particularly in the easier tasks, the generality of the error messages can be misleading. One student attempted to concatenate two string variables with an expression like

```
X$Y$
```

(omitting the concatenation operator "&"), which produced an error message complaining about missing quote marks. Since a very likely explanation for an expression like this that cannot be parsed <u>is</u> missing quote marks, the general message is appropriate. However, this interaction occurred when the student was in a task whose main purpose (reflected in the list of skills)

was to teach about concatenation, and the interpreter could have been aware of this fact if it had access to the skills list. We have found a small number of cases like this, in which a student's error was directly related to the skills in the task but the interpreter presented a nonspecific error message; in those few cases, a link between the parser and the skills list could have produced a more relevant message. In the above example, where the printing skill required by the task is "print string variable expression," the interpreter could process each PRINT statement in a special way, checking specifically for the concatenation operator. The addition of such links would have to be done on a very task-specific basis, since complicated tasks typically involve so many different kinds of statements that overriding the generality of the interpreter could cause equally misleading error messages. (The link, would, however, consist of a call to one of a set of skill-related procedures; it would not be a specific procedure for each task so linked.)

Another more intelligent use of errors would be the addition of specific rules linking structural or execution errors to the student model. When an error such as "missing NEXT following FOR statement" occurs repeatedly during a student's experience with a given task, it might be reasonable to flag the corresponding skill as needing more work even before the student completes (or quits) the task. The link would consist of a check built into the structural analysis procedure, of a form something like: "If a loop error occurs, find the skill in this task that deals with loops, and flag that skill."

An assumption implicit in the procedures that alter the student model is that a student has "learned" a given skill if he successfully completes a task requiring that skill. We have chosen to ignore the number of attempts made before success and the nature of the errors made during the process of developing a correct solution. We do feel that a large number of errors should be forgotten by the instructional program (after the error message is given, of course), because students should be encouraged to experiment with a computer system to help them discover how it works; we do not want students to feel that "typing something wrong" will hold them back. A compromise might be useful: demonstrated difficulty with specific skills, even when followed by success, should be reflected in the student model such that the next task selected will give more practice in the difficult areas. As in the case of syntax errors, the links between the interpreter, the curriculum, and the student model should be relatively few and very specific. Using the above example, it would be important to use the occurrence of the structural error to affect the "FOR . . NEXT" skill only, not the skills that reflect the processing done within the loop.

One of the most useful "hint" features of BIP is the REP command, which displays a flowchart-like representation of the solution to the current task and allows the student to probe the display, expanding the "boxes" of information to see how the structure of the solution program fits together. The REP picture for each task is drawn by the curriculum author when the task is written, not generated automatically; one advantage of this relatively unsophisticated approach is that specific information relating a section of the picture to particular skills can be stored along with it. Then, when a student asks to have that section of the picture expanded, BIP could add to the student model an indication that the student might need more work on the related skills.

Alternative Student Model

In the current working version of BIP, the student model consists of a set of counters associated with each skill, indicating how many times the student had used the skill, how many times he had been successful in the tasks that required it, how many times he had responded with confidence in his own ability in the post-task interview, etc. These counters are used to determine whether or not the student needs more work with the skill at the time a next task is to be selected. We have developed a new six-state model to replace this binary characterization of the state of the skill, where the states are described as follows:

NO--Skill has not been presented, nor have others at its technique level.

N1--Skill has not been presented, but others at its level have been seen.

UO--Skill has been presented but has not been learned.

L3--Lowest level of learning. Skill was required in a task in which the student had difficulty achieving an acceptable solution.

L2--Skill is considered "learned," having been used successfully but in a restricted context of other skills.

L1--Highest learned state. Skill has been used successfully in varied skill contexts.

In addition to indicating whether or not a given skill needs more work, this new model will give BIP more information about the student's knowledge. In particular, it should help in identifying those skills that are the most likely source of a student's difficulty with a given task. Instead of dealing with the entire list of skills when a student chooses to quit a task, for example, it seems reasonable to assume that those skills that were initially least well-learned should be the object of the search in the selection of the next task.

Alternative Form for the CIN Itself

BIP's Curriculum Information Network was, from the beginning, an ad hoc design based on the authors' ideas and feelings about teaching programming. The details of the curriculum description have been changed many times, in response to observed errors, our changing opinions, and our experience with our students. But the fundamental structure of the network has not been changed, since we were interested in doing the best job of individualization that we could with the original design before comparing it with something else. This section describes a new network design we are developing which we hope will eventually provide more insightful curriculum decisions for BIP students.

One shortcoming of BIP's current representation of skills, which groups them by techniques only, is that the inherent semantic relationships between the skills are not utilized. Clearly Skills 3 and 4 are analogous--"print the value of a numeric variable" and "print the value of a string variable." One could infer that if a student knows about printing numeric variables, and about assigning values to string variables, then he would probably have no trouble with the skill of printing the value of a string variable.

Further, inferences can be made about the relative difficulty of certain skills based on value judgments about the difficulty of their components. Thus, if string literals are thought to be harder to use than numeric literals (because of the problems arising from quotes and spaces), then Skill 2 ("print a string literal") can be described as harder than Skill 1 ("print a numeric literal"), since it differs from Skill 1 only with respect to the type of literal being printed. All other pairs of skills with this same minimal difference can have a similar "hardness" relation drawn between them.

Thus, a better task selection algorithm might be devised if, instead of simply grouping skills, we represent them in a notation that points up their similarities and differences, and that allows skills to be functionally related to one another. The new task selection algorithm would have access to much more information than the current one, and could take advantage of inferencing strategies to make a more "intelligent" choice of curriculum.

BASIC Network

As a foundation for specifying relationships between pairs of skills, a semantic network representation of BASIC has been formulated. The nodes of the network were chosen from a consideration of programming concepts in general, and the syntactic and semantic components of the BASIC language in particular. The links between these nodes represent the usual set membership relationships, as well as the subjective pedagogical opinions "harder than," "analogous to," etc. This network embodies a general description of BASIC, rather than a specific description of the BIP tasks and skills; thus, it will remain useful even though the tasks or the skills may be changed.

Figure 5 shows a small piece of the BASIC network. It indicates that there are four kinds (K) of variables: unsubscripted (VAR), subscripted (SUBVAR), numeric (NUMVAR), and string (STRVAR).

The H relation in Figure 5 indicates an opinion that the concept of subscripted variable is generally harder to acquire than that of an unsubscripted one; the bidirectional S and A relations indicate that numeric and string variables are similar in difficulty and analogous to one another. The D relation indicates that the concepts of subscripted variables and the dimension statement are mutually dependent.

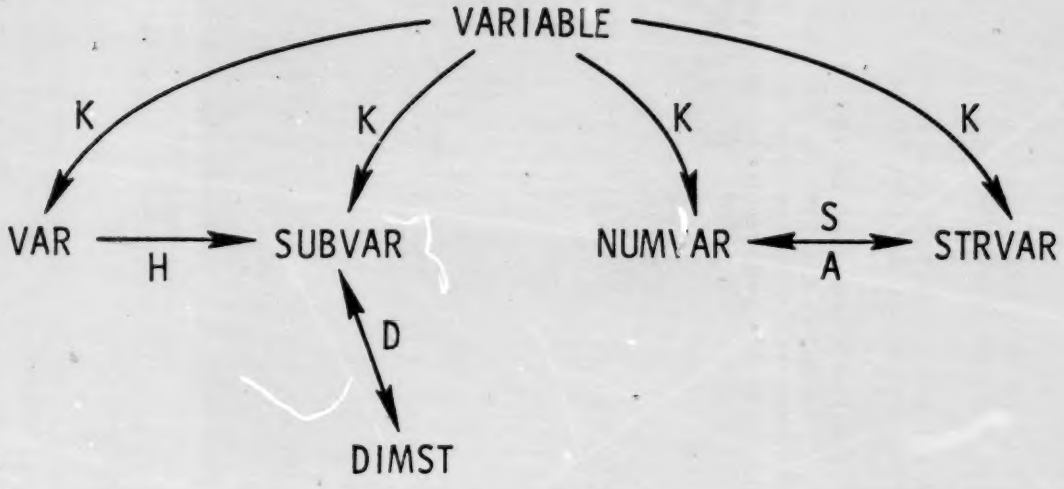

Figure 5. A piece of the BASIC network.

The complete list of link types are:

K--Kind of
C--Component of
[C]--Optional component of
H--Harder than
P--Prerequisite to
S--Of similar difficulty
D--Mutually dependent upon
A--Analogous to
X--Appears analogous to but is different from
I--Identical to (i.e., is the same variable as)

S, D, A, X, and I are always two-way relations; the others are one-way relations in which the direction of the arrow is significant.

In general, the nodes of the net should not be thought of as representing orthogonal classes. In the example in Figure 5, the four classes exhibit completely overlapping membership at the next lower level, since subscripted and unsubscripted variables include both numeric and string types, and numeric and string variables include both subscripted and unsubscripted types. This is partially illustrated in Figure 6, which shows further levels of the subscripted variables (SUBVAR) subnet.

This figure indicates that subscripted variables may be described as being of two particular kinds, numeric (SUBNVAR) and string (SUBSVAR). It also shows that subscripted variables in general consist of two components (C), an unsubscripted variable (VAR) followed by an index. (Since componency is always written from left to right, no overt relationship between component nodes need ever be indicated in the notation.) The particular componency of subscripted numeric (SUBNVAR) and string (SUBSVAR) variables is shown at the next lower level, along with that of the index. The meanings of the names of network nodes are listed in the glossary.

Much of the network is recursive, as is appropriate for a structure that resembles to some extent a BNF for BASIC. The part of the network that describes the simplest skill a student can acquire--printing a numeric literal--is extremely general and recursive. Thus, the print statement (PRINSTS) is seen in Figure 7 as having the components "PRINT" followed by an expression list (EXPRLIST). This expression list may be either a simple expression (EXPR) or a multiple one (EXPRS), which is harder. The possible kinds of expressions are numeric, string, and Boolean, in order of increasing difficulty.

Once the BASIC net was completed, it was represented as a LISP file in a simple property list notation. Each node has as a property the kinds of links that emanate downward from it and to its right; the value of each property is the appropriate list of next levels nodes. Thus, the print statement is represented in LISP as

```
(PRINSTS C ("PRINT" EXPRLIST)),
```

with the SUBVAR node is represented as

```
(SUBVAR K (SUBNVAR SUBVAR) C (VAR INDEX)).
```

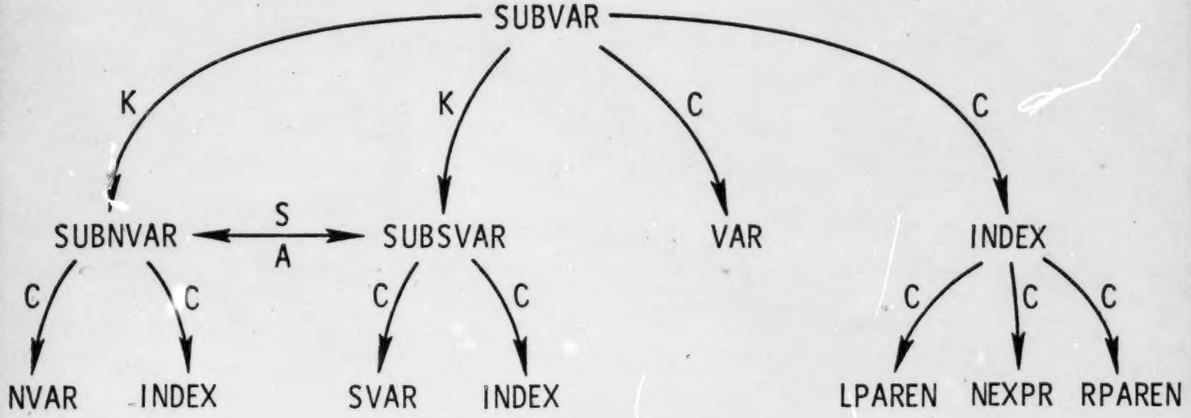

Figure 6. The subscripted variable portion of the BASIC network.

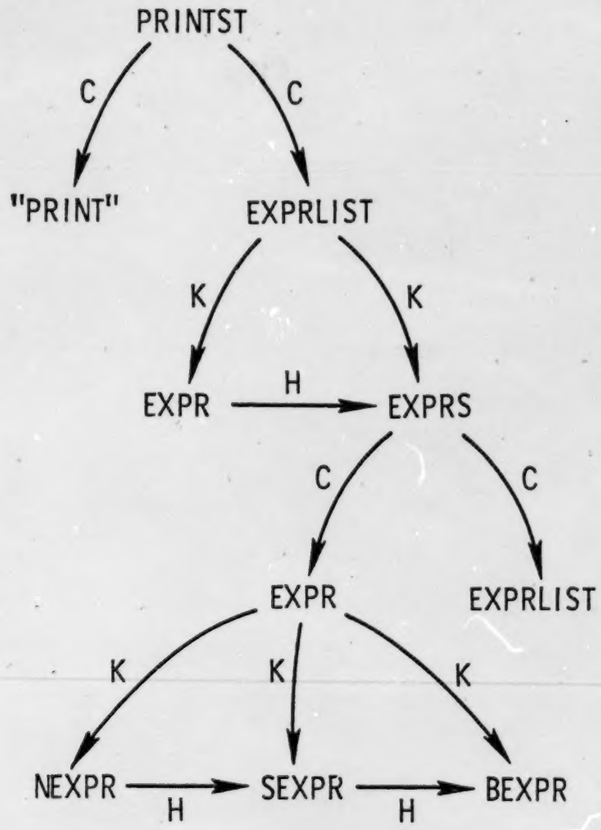

Figure 7. The PRINTST portion of the BASIC network.

This LISP notation provides not only a machine-readable data base (which we can use in the future for program-generated inferences) but also a formal structure that has been helpful in guiding our thinking. The representations of BIP's skills described below were produced by the application of rigorous algorithms to the BASIC net itself. The LIST representation is shown in Appendix B.

Representing the Structure of the Skills

Given a complete BASIC net, we were than able to start devising a notation for representing the 83 skills. Since the BASIC net is general and recursive, the notion of instantiating general nodes in the net with specific nodes for specific skills was introduced. The first ten skills are all printing skills; they involve the use of a simple PRINT statement within a variety of simple string and numeric expressions. They all use the general part of the network shown in Figure 7 and represented in LISP as

(PRINTST C ("PRINT" EXPRLIST)).

Since Skill 1 is "print a numeric literal," it can be represented as

(SK001 (PRINTST (EXPRLIST . NLIT)))

which indicates that, on this occasion, EXPRLIST (an expression list) is to be instantiated by NLIT (a numeric literal), which is one "kind" of EXPRLIST and is itself a node elsewhere in the network.

Using this notation, we were able to represent all the skills in terms of the network and compute the relationships between them. The notation is designed to be descriptive of the relations between the skills and the primitives, and also of the relations between the skills themselves. Thus, skills are defined not only in terms of network nodes, but also in terms of each other.

For example, Skill 2 is "print a string literal," differing from Skill 1 only in the type of literal to be printed. We can represent Skill 2 as

(SK002 (SK001 (EXPRLIST . SLIT)))

indicating that it is the same as Skill 1 except that the expression list is instantiated by a string literal (SLIT). Thus, the notation itself expresses the relationship between the skills. Similarly we could infer, if necessary, that

(SK001 (SK002 (EXPRLIST . NLIT))).

The usefulness of representing skills in terms of each other becomes more apparent as the skills become more complicated. For example, Skill 8 is "print a concatenation of string literals." This can be written as

(SK008 (SK001 (EXPRLIST . (CONCATSEXPR (SEXPR . SLIT)(SEXPR . SLIT))))).

Here several levels of instantiation are indicated. First, Skill 8 is the same as Skill 1 except that its expression list (EXPRLIST) is a concatenated

expression. Then, within the concatenated expression, both of the string expressions are to be instantiated, in this case, by string literals. The dot indicates that instantiation is being expressed within the components list; an open parenthesis indicates a dropping of levels within the network or within another skill expression.

Having written that reasonably complicated description, we can now say that Skill 9, "print a concatenation of variables," is simply Skill 8 using variables instead of literals. That is expressed as

(SK009 (SK008 (SEXPR . SVAR) (SEXPR . SVAR))).

Not only is this succinct, but it also points out clearly the minimal differences between the skills. The complete LISP notation for the structure of the skills is shown in Appendix B.

The Skills Network

Given the LISP notation for the skills, we were then able to group them into "skill sets." A skill set consists of a skill (like Skill 1) that is not described in terms of any other skill, but rather in terms of some network node, and all other skills that are described in terms of that first skill, in a recursive sense. Thus, Skills 1, 2, 8, and 9 above all belong to the same skill set (along with many other skills as well). Ten skill sets were isolated in this way, the two largest ones covering the concepts of assignment by means of the LET statement and printing. A few skills, like the use of the REMARK statement, were not related to any others and were treated independently.

Complex diagrams were drawn for each skill set showing the relations between skills. These relations were derived from the information in the BASIC net, using fairly rigorous algorithms. For example, a hardness link was drawn between Skills 1 and 2, indicating that printing a string literal is generally harder than printing a numeric literal. This link is substantiated by the link in the BASIC net that indicates string literals to be generally harder to use than numeric literals, whether for printing or for other purposes (like assignment, input, or comparison).

The skills net contains the same H, S, and A links found in the BASIC net. In addition, an important new prerequisite link (P) has been added. Thus, Skill 8, "print a concatenation of string literals," has as a prerequisite Skill 2, "print a string literal," since the inexperienced student is considered to need experience with a single literal before he is expected to combine two or more strings. Similarly, Skills 13 and 15 (assignment to a variable via INPUT and READ - DATA respectively) are both given Skill 11 (assignment with the LET statement) as a prerequisite. Clearly, prerequisite links are largely a matter of pedagogic opinion.

The skills net is represented in the same LISP property list notation as that used for the BASIC net. Thus, the fact that Skills 1 and 2 are analogous to one another, and that Skills 2 and 28 are found, by a consideration of the BASIC net, to be harder than Skill 1 is represented as

(SK001 A (SK002) H (SK002 Sk028)).

The complete LISP representation of the relationships between the skills is given in Appendix B.

An Illustration

The following is a description of a mechanism that relies heavily on the explicit relationships between the skills for the "appropriateness" of its task selection. Though we have not yet implemented this design in a full-scale simulation of BIP, the limited simulations we have carried out by hand have been promising, and we intend to evaluate this new scheme, essentially as described here, in the near future.

The network representations of BASIC and the skills present a tremendous array of possibilities for program-generated inferences about the content of the tasks. At some time in the future, we would like to build in some inference-making procedures that might improve BIP's selection of appropriate tasks. At present, however, we will make the following changes to BIP's design, which will not involve the full use of computational mechanisms allowed by the new networks:

1. Replace the "counter" student model with the "states" model described above.

2. Add to the CIN the relationships between skills listed in Appendix B.

3. Implement a new task-selection process based on the modified model and CIN, replacing the technique structure.

The new task-selection design is similar to the old one in that it assembles a set of skills (called the NEED set), on which the student needs work or to which he is ready to proceed, and then locates a task using some of those skills without requiring skills for which he is not yet ready. The process will work as follows, where bracketed words indicate parameters that are expected to be varied experimentally:

1. Identify all skills in states L3 or UO (those that the student has seen but not mastered, or has asked for specifically). If any such skills are found, put them into the NEED set and continue from (4).

2. Locate all skills in states L1 or L2, and find all skills that are ANALOGOUS to them. If any such analogous skills are found, put them into the NEED set and continue from (4).

3. (No skills need remediation, and no immediate analogies are ready to be satisfied.) Identify all unpresented skills whose INVERSE-PREREQUISITES are in states L1 or L2. That is, locate the skills whose prerequisites have been relatively well learned, and put those "ready" skills into the NEED set.

4. Locate all unpresented tasks. Order them by [most] NEEDed skills, then by [fewest] learned L1 or L2) skills.

5. Examine the tasks on the list. If the given task requires any skills whose own prerequisites have not been met, reject the task.

6. If a task is found that does not require such "not ready" skills, present that task. Otherwise, if the list of tasks has been exhausted, find all the unsatisfied prerequisites of the skills in the [last] task examined. [Add] those prerequisite skills to the NEED set and continue from step (4).

We have made a few laborious attempts to follow this design by hand to select a few tasks, and the sequence we have produced is quite reasonable. The manual labor is tremendous, however, and we will soon begin experimenting with at least partially-automated simulations.

The most important difference between this scheme and the current implementation is our reliance on prerequisite and analogy relationships between skills, which are much more explicit than the vague technique groups. Furthermore, the potentially greater precision of the "states" model should allow more accurate distinctions to be made between skills in cases where the student has difficulty or quits the task. The "hardness" links between the skills will probably be added to the scheme, mainly to provide further inferences for identifying NEEDed skills, once we verify the general usefulness of the design.

CONCLUSIONS AND RECOMMENDATIONS

We feel that the Curriculum Information Network shows great promise as a fundamental structure for CAI curriculums in technical problem-solving areas. One topic worth a brief note at this point is that of CAI "author languages." As reflected in our discussion of curriculum-based branching CAI, we are not enthusiastic about attempts to make it trivially easy for a subject matter expert to design his own instructional material within the framework of a lesson-oriented branching CAI system. The limitations imposed on dynamic runtime individualization of the presentation of material are simply too great. The purely generative approach, on the other hand, makes it impossible for a nonprogrammer to contribute very much to the design of a course, which obviously limits the author pool to an initiated few.

Writing curriculum for the BIP course, by contrast, is easy. The author writes the text, hints, and model solution more or less as he pleases, and then constructs the list of skills for the task with reference to the list of skills and their numbers. We do not mean to underestimate the effort involved in constructing the program itself--a job requiring cooperation between sophisticated programmers and equally sophisticated subject matter experts. However, once such a program exists for a given course, the CIN provides an extremely flexible structure within which curriculum may be added by nonexperts. In applications such as Navy training, a system like BIP's might be constructed for some technical subject area that is taught at a number of different installations. Individual instructors at each site could then be encouraged to produce specialized curriculum sections, tailoring the content of the course to their own requirements.

REFERENCES

Atkinson, R. C., Fletcher, J. D., Lindsay, E. J., Campbell, J. O., & Barr, A. Computer-assisted instruction in initial reading (Tech. Rep. 207). Stanford, CA: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

Barr, A. A survey of artificial intelligence in computer-assisted instruction (Unpublished manuscript). Stanford, CA: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1976.

Barr, A., Beard, M., & Atkinson, R. C. A rationale and description of a CAI program to teach the BASIC programming language. Instructional Science, 1975 4, 1-31. (a)

Barr, A., Beard, M., & Atkinson, R. C. The computer as a tutorial laboratory: The Stanford BIP project (Tech. Rep. 260). Stanford, CA: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1975. (b)

Beard, M., Lorton, P., Searle, B., & Atkinson, R. C. Comparison of student performance and attitude under three lesson-selection strategies in computer-assisted instruction (Tech. Rep. 222). Stanford, CA: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

Bork, A. Learning with computers--today and tomorrow. In O. Lecarme & R. Lewis (Eds.), Computers in education (IFIP 2nd World Conference). Amsterdam: North Holland, 1975.

Brown, J. S., Burton, R. R., & Bell, A. An intelligent CAI system that reasons and understands (BBN Rep. 2790). Cambridge, MA: Bolt, Beranek, and Newman, Inc. 1974.

Carbonell, J. R. AI in CAI: An artificial intelligence approach to computer-aided instruction. IEEE Transactions on Man-Machine Systems, 1970, MMS-11, 190-202.

Collins, A., & Grignetti, M. Intelligent CAI (BBN Rep 3181). Cambridge, MA: Bolt, Beranek, and Newman, Inc. 1975.

Collins, A. M., Passafiume, J. J., Gould, L., & Carbonell, J. R. Improving interactive capabilities in computer-assisted instruction (BBN Rep. 2631). Cambridge, MA: Bolt, Beranek, and Newman, Inc. 1973.

Danforth, D. G., Rogasa, D. R., & Suppes, P. Learning models for realtime speech recognition (Tech. Rep 223). Stanford, CA: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1974.

Friend, J. Computer-assisted instruction in programming: A curriculum description (Tech. Rep. 211). Stanford, CA: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

Friend, J. Programs students write (Tech. Rep. 257). Stanford, CA: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1975.

Goldberg, A. Computer-assisted instruction: The application of theorem-proving to adaptive response analysis (Tech. Rep. 203). Stanford, CA: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

Goldstein, I. Summary of MYCROFT: A system for understanding simple picture programs. Artificial Intelligence, 1975, 6, 249-288.

Grignetti, M. C., Hausmann, C., & Gould, L. An "intelligent" on-line assistant and tutor: NLS-SCHOLAR. Proceedings of the National Computer Conference, Anaheim, CA, 1975, 775-781.

Kimball, R. B. Self-optimizing computer-assisted tutoring: Theory and practice (Tech. Rep. 206). Stanford, CA: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

Koffman, E. B. A generative CAI tutor for computer science concepts. Proceedings AFIPS 1972 Spring Joint Computer Conference, 379-389.

Koffman, E. B., & Blount, S. E. Artificial intelligence and automatic programming in CAI. Artificial Intelligence, 1975, 6, 215-234.

Ruth, G. Analysis of algorithm implementations (MAC TR-130). Cambridge, MA: Massachusetts Institute of Technology, 1974.

Sanders, W., Benbassat, G., & Smith, R. L. Speech synthesis for computer-assisted instruction: The MISS system and its applications. SIGSCE Bulletin, 1976, 8, 200-211.

Smith, R. L., & Blaine, L. A generalized system for university mathematics instruction. SIGSCE Bulletin, 1976, 8, 280-298.

Smith, R. I., Graves, H., Blaine, L. H., & Marinov, V. G. Computer-assisted axiomatic mathematics: Informal rigor. In O. Lerarme & R. Lewis (Eds.), Computers in education (IFIP 2nd World Conference). Amsterdam: North Holland, 1975.

Van Campen, J. A. Project for application of learning theory to problems of second language acquisition with particular reference to Russian. Report to U.S. Office of Education, Contract No. DEC-0-8-001209-1806, 1970.

REFERENCE NOTES

1. Beard, M. H., & Barr, A. V. The BASIC instructional program student manual (NPRDC Special Rep. 77-2). San Diego: Navy Personnel Research and Development Center, October 1976.

2. Dageforde, M. L., & Beard, M. H. The BASIC instructional program: Supervisor's manual (NPRDC Tech. Note 78-10). San Diego: Navy Personnel Research and Development Center, April 1978.

3. Dageforde, M. L. The BASIC instructional program: Conversion into MAINSAIL language (NPRDC Tech. Note 78-11). San Diego: Navy Personnel Research and Development Center, April 1978.

4. Dageforde, M. L. The BASIC instructional program: System documentation (NPRDC Tech. Note 78-12). San Diego: Navy Personnel Research and Development Center, April 1978.

5. Dageforde, M. L., Beard, M., & Barr, A. V. The BASIC instructional program student manual: MAINSAIL conversion (NPRDC Tech. Note 78-9). San Diego: Navy Personnel Research and Development Center, April 1978.

GLOSSARY

ADD	addition operator
ADDSUB	additional and subtraction operators
ARITH	arithmetic operator
ARITHNEXPR	arithmetic expression
ASSIGNST	assignment statement
BASICPROGRAM	BASIC program
BEXPR	Boolean expression, either simple or complex
BOOLEREL	complex Boolean expression with Boolean operator(s)
BOOLOP	Boolean operator (AND, OR, or NOT)
CHARACTER	any keyboard symbol known to BASIC
COMP	GT and LT
COMPLARITH	complete arithmetic expression
COMPLEXPR	complex expression (operator - either string or numeric)
CONCAT	concatenation operator
CONCATSEXPR	string expression with concatenation operator(s)
CONDITIONAL	conditional branching
CONTROLST	control statement
DATAST	DATA statement
DIGIT	0 through 9
DIGITS	one or more DIGIT
DIMST	dimension statement
DIV	division operator
ENDST	END statement
ENDSTATEMENT	the END statement, including its line number
EQCOMP	LE and GE
EQUAL	EQ and NEQ
EXP	exponentiation operator
EXPR	expression (either string or numeric)
EXPRLIST	expression list
EXPRS	multiple expressions (in a PRINT statement)
FORNEXTST	combination of FOR statement and NEXT statement
FORST	FOR statement
FUNCNEXPR	function
GOTOST	GOTO statement
IFTHENST	IF-THEN statement
INDEX	index part of a subscripted variable
INNAME	name of an in statement - either INPUT or READ
INPUTST	INPUT statement
INST	in statement
INT	integer function
INTEGER	integer number (positive, negative, or zero)
IOST	I/O statement

LETST	LET statement
LITERAL	literal (either string or numeric)
LITLIST	literal list (for use with DATA statement)
LITS	multiple literals (for use with DATA statement)
MULT	multiplication operator
MULTDIV	multiplication and division operators
NEG	negative integer
NEXPR	numeric expression
NEXTST	NEXT statement
NLIT	numeric literal
NOSTEP	nil (missing) step expression (in a FOR statement)
NOTONE	positive integer, not one
NREL	simple Boolean expression relating numeric expressions
NUMLET	numeric LET statement
NUMVAR	numeric variable (either simple or subscripted)
NVAR	simple numeric variable
OPERATOR	operator - either arithmetic, string, or Boolean
POS	positive integer
PRINTST	PRINT statement
READATAST	combination of READ statement and DATA statement
READST	READ statement
REAL	floating point number
RELOP	relational operator - EQ, NEQ, GT, LT, LE, GE
REMST	remark statement
RND	random number function
SEXPR	string expression
SIMARITH	simple arithmetic expression
SIMNEXPR	simple numeric expression
SIMPLEXPR	simple expression (no operator - either string or numeric)
SIMREL	simple Boolean expression with one relational operator
SIMSEXPR	simple string expression
SLIT	string literal
SQR	square root function
SREL	simple Boolean expression relating string expressions
STATEMENTLINES	BASIC statements, including line numbers
STATEMENTS	BASIC statements, exclusive of line numbers
STEP	overt step expression (in a FOR statement)
STEPEXPR	step expression (in a FOR statement - may be nil)
STOPST	STOP statement
STRLET	string LET statement
STRVAR	string variable (either simple or subscripted)
SUB	subtraction operator
SUBNVAR	subscripted numeric variable
SUBSVAR	subscripted string variable
SUBVAR	subscripted variable (either string or numeric)
SVAR	simple string variable
SYMBOL	CHARACTER which is not a letter, digit, or a space

TEXT	arbitrary text (as in a REMARK statement)
UNCONDITIONAL	unconditional branching
VAR	simple variable (either string or numeric)
VARIABLE	all variables - string and numeric, simple and subscripted
VARIABLES	multiple variables (for use with INPUT or READ statement)
VARLIST	variable list (for use with INPUT or READ statement)

APPENDIX A

THE TECHNIQUE GROUPS AND THE SKILLS

THE TECHNIQUE GROUPS AND THE SKILLS

Technique 1. Simple output--first programs.

1 Print numeric literal
2 Print string literal
5 Print numeric expression [operation on literals]
8 Print string expression [concatanation of literals]

Technique 2. Variables--assignment.

3 Print value of numeric variable
4 Print value of string variable
6 Print numeric expression [operation on variables]
7 Print numeric expression [operation on literals and variables]
9 Print string expression [concatanation of variables]
10 Print string expression [concatanation of variable and literal]
11 Assign value to a numeric variable [literal value]
12 Assign value to a string variable [literal value]

Technique 3. More complicated assignment.

34 Assign to a string variable [value of an expression]
35 Assign to a numeric variable [value of an expression]
69 Re-assignment of string variable (using its own value)
70 Re-assignment of numeric variable (using its own value)
82 Assign to numeric variable the value of another variable
83 Assign to string variable the value of another variable

Technique 4. More complicated output.

28 Multiple print [string literal, numeric variable]
29 Multiple print [string literal, numeric variable expression]
30 Multiple print [string literal, string variable]
74 Multiple print [string literal, string variable expression]

Technique 5. Interactive programs--INPUT from user--using DATA.

13 Assign numeric variable by -INPUT-
14 Assign string variable by -INPUT-
15 Assign numeric variable by -READ- and -DATA-
16 Assign string variable by -READ- and -DATA-
55 The REM statement

Technique 6. More complicated input.

17 Multiple values in -DATA- [all numeric]
18 Multiple values in -DATA- [all string]
19 Multiple values in -DATA- [mixed numeric and string]
22 Multiple assignment by -INPUT- [numeric variables]
23 Multiple assignment by -INPUT- [string variables]
24 Multiple assignment by -INPUT- [mixed numeric and string]
25 Multiple assignment by -READ- [numeric]
26 Multiple assignment by -READ- [string]
27 Multiple assignment by -READ- [mixed numeric and string]

Technique 7. Branching--program flow.

36 Unconditional branch (-GOTO-)
37 Interrupt execution

Technique 8. Boolean expressions.

38 Print Boolean expression [relation of string literals]
39 Print Boolean expression [relation of numeric literals]
40 Print Boolean expression [relation of numeric literal and variable]
41 Print Boolean expression [relation of string literal and variable]
75 Boolean operator -AND-
76 Boolean operator -OR-
77 Boolean operator -NOT-

Technique 9. IF statements—conditional standards.

42 Conditional branch [compare numeric variable with numeric literal]
43 Conditional branch [compare numeric variable with expression]
46 Conditional branch [compare two numeric variables]
47 Conditional branch [compare string variable with string literal]
48 Conditional branch [compare two string variables]
59 The -STOP- statement

Technique 10. Hand-made loops--iteration.

44 Conditional branch [compare counter with numeric literal]
45 Conditional branch [compare counter with numeric variable]
49 Initialize counter variable with a literal value
50 Initialize counter variable with the value of a variable
53 Increment the value of a counter variable
54 Decrement the value of a counter variable

Technique 11. Using loops to accumulate.

51 Accumulate successive values into numeric variable
52 Accumulate successive values into string variable
71 Calculating complex expressions [numeric literal and variable]
78 Initialize numeric variable (not counter) to literal value
79 Initialize numeric variable (not counter) to value of a variable
80 Initialize string variable to literal value
81 Initialize string variable to the value of another variable

Technique 12. Using "dummy" value to signify end of data.

20 Dummy value in -DATA- statement [numeric]
21 Dummy value in -DATA- statement [string]

Technique 13. BASIC functionals.

56 The -INT- function
57 The -RND- function
58 The -SQR- function

Technique 14. FOR...NEXT loops.

61 FOR . NEXT loops with literal as final value of index
62 FOR . NEXT loops with variable as final value of index
63 FOR . NEXT loops with positive step size other than 1
64 FOR . NEXT loops with negative step size

Technique 16. Arrays.

31 Assign element of string array variable by -INPUT-
32 Assign element of numeric array variable by -INPUT-
33 Assign element of numeric array variable [value is also a variable]
60 The -DIM- statement
65 String array using numeric variable as index
66 Print value of an element of a string array variable
67 Numeric array using numeric variable as index
68 Print value of an element of a numeric array variable

Technique 16. Nesting loops (one loop inside another).

72 Nesting loops
73 Subroutines (-GOSUB- and friends)

APPENDIX B

THE LISP NOTATION FOR THE BASIC NETWORK, THE SKILLS STRUCTURE, AND THE SKILLS RELATIONSHIPS

1. THE LISP NOTATION FOR THE BASIC NETWORK:

```
(BASICPROGRAM C ((STATEMENTLINES . OPT) ENDSTATEMENT))
(STATEMENTLINES C (LINENUM (STATEMENTS . OPT)))
(ENDSTATEMENT C (LINENUM ENDST))
(STATEMENTS K (REMST IOST ASSIGNST CONTROLST DIMST))
(ENDST C (%END%))
(REMST C (%REM% (TEXT . OPT)))
(IOST K (PRINTST INST))
(INST H (PRINTST LETST) C (INNAME VARLIST) K (INPUTST READATAST))
(ASSIGNST K (INST LETST))
(DIMST C (%DIM% VAR NEXPR) D (SUBVAR))
(CONTROLST K (UNCONDITIONAL CONDITIONAL))
(CONDITIONAL H (UNCONDITIONAL) K (IFTHENST FORNEXTST))
(UNCONDITIONAL K (ENDST GOTOST STOPST))
(STOPST H (GOTOST) C (%STOP%))
(FORST D (NEXTST) C (%FOR% (NUMVAR . 1) %FROM% NEXPR %TO% NEXPR STEPEXPR))
(GOTOST C (%GOTO% LINENUM) H (ENDST))
(LETST K (NUMLET STRLET) C ((%LET% . OPT) VARIABLE GETS EXPR))
(STEPEXPR K (NOSTEP STEP))
(FORNEXTST H (IFTHENST) C (FORST NEXTST))
(NEXTST D (FORST) C (%NEXT% (NUMVAR . 1)))
(NUMLET C ((%LET% . OPT) NUMVAR GETS NEXPR))
(STRLET C ((%LET% . OPT) STRVAR GETS SEXPR) H (NUMLET))
(NOSTEP C (NILL))
(STEP C (%STEP% NEXPR) H (NOSTEP))
(IFTHENST C (%IF% BEXPR %THEN% LINENUM))
(PRINTST C (%PRINT% EXPRLIST))
(EXPRLIST K (EXPR EXPRS))
(EXPRS H (EXPR) C (EXPR EXPRLIST))
(EXPR K (SIMPLEXPR COMPLEXPR NEXPR SEXPR BEXPR))
(SIMPLEXPR K (SIMNEXPR SIMSEXPR))
(COMPLEXPR H (SIMPLEXPR) C (SIMPLEXPR OPERATOR EXPR))
(SIMSEXPR H (SIMNEXPR) K (SLIT STRVAR) A (SIMNEXPR))
(SEXPR K (SIMSEXPR CONCATSEXPR) H (NEXPR))
(CONCATSEXPR C (SEXPR CONCAT SEXPR) H (SIMSEXPR) A (SIMARITH))
(SIMNEXPR K (NLIT NUMVAR) A (SIMSEXPR))
(NEXPR K (SIMNEXPR ARITHNEXPR FUNCNEXPR))
(ARITHNEXPR K (SIMARITH COMPLARITH) H (SIMNEXPR))
(SIMAPITH C (SIMNEXPR ARITH SIMNEXPR) A (CONCATSEXPR))
(COMPLARITH C (SIMARITH ARITH NEXPR) H (SIMARITH))
(FUNCNEXPR K (SQR INT RND) H (SIMNEXPR))
(SQR C (%SQR% NEXPR))
(INT C (%INT% NEXPR) H (SQR) S (RND))
(RND C (%RND%) S (INT))
(BEXPR K (SIMREL BOOLEREL) H (SEXPR))
(BOOLEREL C (BEXPR BOOLOP BEXPR) H (SIMREL))
(SIMREL K (NREL SREL))
(SREL C (SEXPR RELOP SEXPR) H (NREL))
(NREL C (NEXPR RELOP NEXPR))
(INNAME K (%READ% %INPUT%))
(INPUTST C (%INPUT% VARLIST))
(VARLIST K (VARIABLE VARIABLES))
(VARIABLES H (VARIABLE) C (VARIABLE VARLIST))
(READATAST C (READST DATAST) H (INPUTST))
(READST C (%READ% VARLIST) D (DATAST) S (DATAST))
(DATAST C (%DATA% LITLIST) D (READST) S (READST))
(LITLIST K (LIT LITS))
(LITS C (LIT LITLIST) H (LIT))
(OPERATOR K (ARITH CONCAT RELOP BOOLOP))
```

```
(EXPR K (ADDSUB MULTDIV EXP))
(EXP H (MULTDIV) C (%/^%))
(MULTDIV H (ADDSUB) K (MULT DIV))
(DIV H (MULT) C (%//%))
(MULT C (%/*%))
(ADDSUB K (ADD SUB))
(SUB H (ADD) C (%/-%))
(ADD C (%/+%) A (CONCAT))
(CONCAT C (%/&%) A (ADD) H (ADD))
(BOOLOP K (AND OR NOT) H (RELOP))
(OR H (AND) C (%OR%))
(AND C (%AND%))
(TEXT C (CHARACTER (TEXT . OPT)))
(RELOP K (EQUAL COMP EQCOMP))
(EQCOMP H (COMP) K (GE LE))
(LE S (GE) C (%/</=%))
(GE S (LE) C (%/>/=%))
(COMP K (GT LT) H (EQUAL))
(GT S (LT) C (%/>%))
(LT S (GT) C (%/<%))
(EQUAL K (EQ NEQ))
(NEQ H (EQ) C (%/>/<%))
(EQ C (%/=%))
(VARIABLE K (VAR SUBVAR NUMVAR STRVAR) H (LITERAL))
(SUBVAR D (DIMST) K (SUBNVAR SUBSVAR) H (VAR) C (VAR INDEX))
(VAR K (NVAR SVAR))
(SVAR A (NVAR) S (NVAR))
(NVAR A (SVAR) S (SVAR))
(SUBSVAR S (SUBNVAR) A (SUBNVAR) C (SVAR INDEX) H (SVAR))
(SUBNVAR C (NVAR INDEX) S (SUBSVAR) A (SUBSVAR) H (NVAR))
(INDEX C (%/(% NEXPR %/)%))
(NUMVAR K (NVAR SUBNVAR) S (STRVAR) A (STRVAR) H (NLIT))
(STRVAR K (SVAR SUBSVAR) S (NUMVAR) A (NUMVAR) H (SLIT))
(LITERAL K (NLIT SLIT))
(SLIT C (QUOTE TEXT QUOTE) H (NLIT) A (NLIT))
(NULL H (SPACE) A (ZERO))
(SPACE H (SYMBOLS) C (%/ %))
(CHARACTER K (LETTER DIGIT SYMBOL SPACE NULL))
(SYMBOL H (DIGIT))
(DIGIT H (LETTER) X (POS) K (%0% %1% %2% %3% %4% %5% %6% %7% %8% %9%))
(NLIT A (SLIT) K (INTEGER REAL))
(REAL H (INTEGER))
(INTEGER K (POS NEG ZERO))
(NEG H (ZERO) C (%/-% DIGITS))
(DIGITS C (DIGIT (DIGITS . OPT)))
(POS X (DIGIT) K (ONE NOTONE) C ((%+% . OPT) DIGITS))
(ZERO H (POS) A (NULL) C (%0%))
(NOTONE H (ONE))
(ONE C (%1%))
(NOT H (OR) C (%NOT%))
```

2. THE LISP NOTATION FOR THE SKILLS STRUCTURE:

```
(SK001 (PRINTST (EXPRLIST . NLIT]
(SK002 (SK001 (EXPRLIST . SLIT]
(SK003 (SK001 (EXPRLIST . NVAR]
(SK004 (SK001 (EXPRLIST . SVAR]
(SK005 (SK001 (EXPRLIST . (SIMARITH (SIMNEXPR . NLIT)(SIMNEXPR . NLIT]
(SK006 (SK005 (EXPRLIST . (SIMARITH (SIMNEXPR . NVAR)(SIMNEXPR . NVAR]
(SK007 (SK006 (EXPRLIST . (SIMARITH (SIMNEXPR . NLIT)
(SK008 (SK001 (EXPRLIST . (CONCATSEXPR (SEXPR . SLIT)(SEXPR . SLIT]
(SK009 (SK008 (SEXPR . SVAR)(SEXPR . SVAR]
(SK010 (SK009 (SEXPR . SLIT]
(SK011 (NUMLET (NUMVAR . NVAR)(NEXPR . NLIT]
(SK012 (STRLET (STRVAR . SVAR)(SEXPR . SLIT]
(SK013 (INPUTST (VARLIST . NVAR]
(SK014 (SK013 (VARLIST . SVAR]
(SK015 (READATAST (READST (VARLIST . NVAR))(DATAST (LITLIST . NLIT]
(SK016 (SK015 (READST (VARLIST . SVAR))(DATAST (LITLIST . SLIT]
(SK017 (DATAST (LITLIST . (LITS (LIT . NLIT)(LITLIST . NLIT]
(SK018 (SK017 (LIT . SLIT)(LITLIST . SLIT]
(SK019 (SK018 (LIT . NLIT]
(SK020 (SK017]
(SK021 (SK018]
(SK022 (INST (INNAME . %INPUT%)(VARLIST . (VARIABLES (VARIABLE .
                                                    NVAR)(VARLIST . NVAR]
(SK023 (SK022 (VARIABLE . SVAR)(VARLIST . SVAR]
(SK024 (SK023 (VARIABLE . NVAR]
(SK025 (SK022 (INNAME . %READ%]
(SK026 (SK023 (INNAME . %READ%]
(SK027 (SK024 (INNAME . %READ%]
(SK028 (SK001 (EXPRLIST . (EXPRS (EXPR . SLIT)(EXPRLIST . NVAR]
(SK029 (SK028 (EXPR . SLIT)(EXPRLIST . SIMARITH]
(SK030 (SK028 (EXPR . SLIT)(EXPRLIST . SVAR]
(SK031 (SK013 (VARLIST . SUBSVAR]
(SK032 (SK013 (VARLIST . SUBNVAR]
(SK033 (SK011 (NUMVAR . SUBNVAR)(NEXPR . NVAR]
(SK034 (SK012 (SEXPR . CONCATSEXPR]
(SK035 (SK011 (NEXPR . SIMARITH]
(SK036 (GOTOST]
(SK037 (CTRLG]
(SK038 (SK001 (EXPRLIST . (SREL (SEXPR . SLIT)(SEXPR . SLIT]
(SK039 (SK038 (EXPRLIST . (NREL (NEXPR . NLIT)(NEXPR . NLIT]
(SK040 (SK039 (NEXPR . NVAR]
(SK041 (SK038 (SEXPR . SVAR]
(SK042 (IFTHENST (BEXPR . (NREL (NEXPR . NLIT)(NEXPR . NVAR]
(SK043 (SK042 (NEXPR . SIMARITH]
(SK044 (SK042]
(SK045 (SK046]
(SK046 (SK042 (NEXPR . NVAR]
(SK047 (SK042 (BEXPR . (SREL (SEXPR . SLIT)(SEXPR . SVAR]
(SK048 (SK047 (SEXPR . SVAR]
(SK049 (SK011]
(SK050 (SK082]
(SK051 (SK070 (ARITH . %+%]
(SK052 (SK069]
(SK053 (SK051]
(SK054 (SK070 (ARITH . %-%]
(SK055 (REMST]
(SK056 (INT]
(SK057 (RND]
```

```
(SK058 (SQR]
(SK059 (STOPST]
(SK060 (DIMST (NEXPR . NOTONE]
(SK061 (FORNEXTST (FORST (NUMVAR . (NVAR . 1))(NEXPR . NLIT)(NEXPR .
                         NLIT)(STEPEXPR . NULL))(NEXTST (NUMVAR . (NVAR . 1]
(SK062 (SK061 (NEXPR . NLIT)(NEXPR . NVAR]
(SK063 (SK061 (NEXPR . SIMNEXPR)(NEXPR . SIMNEXPR)(STEPEXPR . (STEP (NEXPR
                                                                  . NOTONE]
(SK064 (SK063 (STEPEXPR . (STEP (NEXPR . NEG]
(SK065 (SUBSVAR (NEXPR . NVAR]
(SK066 (SK001 (EXPRLIST . SUBSVAR]
(SK067 (SUBNVAR (NEXPR . NVAR]
(SK068 (SK001 (EXPRLIST . SUBNVAR]
(SK069 (SK012 (STRVAR . (SVAR . 1))(SEXPR . (CONCATSEXPR (SEXPR . (SVAR .
                                                                    1]
(SK070 (SK011 (NUMVAR . (NVAR . 1))(NEXPR . (SIMARITH (SIMNEXPR . (NVAR .
                                                                    1]
(SK071 (COMPLARITH]
(SK074 (SK028 (EXPR . SLIT)(EXPRLIST . CONCATSEXPR]
(SK075 (BOOLEREL (BOOLOP . %AND%]
(SK076 (SK075 (BOOLOP . %OR%]
(SK077 (SK075 (BOOLOP . %NOT%]
(SK078 (SK011]
(SK079 (SK082]
(SK080 (SK012]
(SK081 (SK083]
(SK082 (SK011 (NEXPR . NVAR]
(SK083 (SK012 (SEXPR . SVAR]
```

3. FOR THE SKILLS RELATIONSHIPS:

```
(SK001 A (SK002) H (SK002 SK028))
(SK002 H (SK003 SK023 SK005))
(SK003 S (SK004) A (SK004) H (SK028 SK068))
(SK004 H (SK005 SK066 SK030))
(SK005 H (SK029 SK008 SK006) A (SK008) P (SK001))
(SK008 H (SK009 SK074) P (SK002))
(SK009 H (SK010) P (SK004))
(SK028 A (SK030) S (SK030) P (SK003 SK002))
(SK030 H (SK029) P (SK004 SK002))
(SK029 H (SK074) A (SK074) P (SK005 SK002))
(SK066 A (SK068) S (SK068) P (SK004))
(SK006 H (SK007 SK009) P (SK003) A (SK009))
(SK007 H (SK039 SK068 SK010) P (SK001 SK003) A (SK010))
(SK039 H (SK038 SK040) A (SK038) P (SK001))
(SK038 H (SK041) P (SK002))
(SK040 H (SK041) A (SK041) P (SK001 SK003))
(SK010 H (SK074 SK039) P (SK002 SK004))
(SK011 A (SK012) H (SK012 SK033 SK049 SK082) P (SK001))
(SK082 H (SK035 SK079) A (SK083) S (SK083) P (SK011))
(SK035 H (SK051 SK034) A (SK034) P (SK005 SK011))
(SK051 H (SK053 SK054 SK069) A (SK069) P (SK006 SK035))
(SK054 H (SK070) P (SK006 SK035))
(SK053 A (SK052) H (SK052) P (SK051))
(SK079 A (SK081) S (SK081) H (SK050) P (SK082))
(SK012 H (SK080 SK083) P (SK002))
(SK083 H (SK034 SK081) P (SK012))
(SK034 H (SK069) P (SK008 SK012))
(SK069 H (SK052) P (SK009 SK034))
(SK049 H (SK080) A (SK080) P (SK011))
(SK081 H (SK050) A (SK050) P (SK083))
(SK042 H (SK044 SK046 SK047) A (SK047) P (SK036 SK040 SK003))
(SK048 H (SK047) P (SK004 SK036 SK038))
(SK046 H (SK043  SK048 SK045) A (SK048) P (SK003 SK036 SK039))
(SK044 H (SK045) A (SK045) P (SK049 SK042))
(SK043 H (SK047 SK075 SK061) P (SK036 SK040 SK003 SK005))
(SK045 H (SK047) P (SK046))
(SK047 H (SK075 SK061) P (SK038 SK004 SK036))
(SK075 H (SK076) P (SK039))
(SK076 H (SK077) P (SK039))
(SK061 H (SK062) P (SK042))
(SK062 H (SK063) P (SK042))
(SK063 H (SK064) P (SK061))
(SK013 A (SK014 SK015) H (SK015 SK022) S (SK014) P (SK011))
(SK014 H (SK032 SK023 SK024) P (SK012))
(SK031 S (SK032) A (SK032) P (SK014))
(SK022 S (SK023) A (SK023 SK025) H (SK025) P (SK013))
(SK023 H (SK024 SK026) A (SK026) P (SK014))
(SK024 H (SK027) A (SK027) P (SK013 SK014))
(SK025 S (SK026) A (SK026) P (SK015))
(SK026 H (SK027) P (SK016))
(SK015 H (SK016 SK017) A (SK016) P (SK011))
(SK017 H (SK018 SK020) A (SK018) P (SK015))
(SK018 H (SK019 SK021) P (SK016))
(SK016 H (SK026 SK018) P (SK012))
(SK020 H (SK021) A (SK021) P (SK017))
(SK074 P (SK008 SK002))
(SK068 P (SK003 SK044))
(SK056 H (SK058) P (SK035))
(SK033 P (SK044 SK082))
```

```
(SK021 P (SK018))
(SK080 P (SK012))
(SK081 P (SK083))
(SK050 P (SK082))
(SK052 P (SK069))
(SK041 P (SK002 SK004))
(SK070 P (SK006 SK035))
(SK064 P (SK061))
(SK058 H (SK057) P (SK035))
(SK057 P (SK035))
(SK022 P (SK013))
(SK077 P (SK039))
(SK068 P (SK044))
(SK059 P (SK042))
(SK065 A (SK057) S (SK067) P (SK044 SK012))
(SK067 P (SK044 SK011))
(SK071 P (SK035))
```